AMATH 582 Homework 2

William Livernois (willll@uw.edu)

February 10, 2020

Abstract

Time-frequency analysis using the Gabor Transform was used to analyze clips from two popular rock and roll songs, *Sweet Child O' Mine* by Guns N' Roses and *Comfortably Numb* by Pink Floyd. Discrete window sampling was used to create a spectrogram of each song in MATLAB and frequency filtering was used to determine the bass and melody in the clips.

1 Introduction and Overview

Time-frequency analysis is a very important tool for all kinds of signal processing, especially audio data. Music is generated by timing sound from an instrument, usually tuned to specific audio frequencies, creating a series of superimposed wave-forms that can cause your brain to spray the good chemical. The Fourier transform is a great tool for extracting all frequency information from a data sample, deconstructing a signal into a sum of plane waves. However, when the Fourier Transform is used across the whole data set all time information is lost. To mitigate this a sliding window can be used to sample a subsection of data, increasing temporal resolution at the cost of spectral resolution. This idea, called Gabor's uncertainty principle, can be summarized as

$$\Delta f \Delta t \geq \frac{1}{2} \tag{1}$$

setting an upper bound to the certainty of time and frequency[2]. The Gabor Transform, also called the short-time Fourier transform, minimizes the uncertainty in time and frequency as a simple method to generate spectral data as a function of time. This idea can be visualized in Figure 1, and the Gabor transform can be applied to music for note identification.

The Gabor transform was applied to the opening of *Sweet Child O' Mine* by Guns N' Roses to identify the notes in the guitar riff and the guitar solo in *Comfortably Numb* by Pink Floyd to analyze the bass and melody components. These chosen songs have what some might call "sick" guitar riffs, creating a clear frequency signature that can be visualized in frequency space.

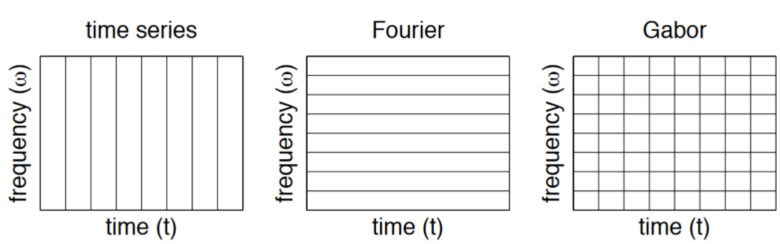

Figure 1: A diagram comparing the resolution in time and frequency of the original signal, Fourier Transformed signal, and Gabor transformed signal [3]

2 Theoretical Background

The Gabor transform uses a sliding window function to collect time/frequency information. The mathematical definition is given by

$$G[f](t,\omega) = \int_{-\infty}^{\infty} f(t)g(t'-t)e^{-i\omega t'}dt' \tag{2}$$

where the $f(t)$ is the continuous signal and $g(t-t')$ represents the window function. Applying this to a discrete data set can be simplified into two steps: multiplying the data set by $g(t-t')$ and applying a discrete Fourier transform. In the implementation used here $g(t-t')$ was defined as a localized Gaussian function defined along a sub-interval of the dataset:

$$g(t-t') = \begin{cases} \exp\left(-10\frac{(t-t')^2}{w^2}\right) & t' \in [t-\frac{w}{2}, t+\frac{w}{2}] \\ 0 & t' \notin [t-\frac{w}{2}, t+\frac{w}{2}] \end{cases} \tag{3}$$

with the interval defined by the width parameter w. After spectrogram manipulation (applying filters, setting cutoffs) transforming the two-dimensional representation back to a one-dimensional signal just required reversing the steps: inverse Fourier transform, dividing by the filter, and patching together in the main signal.

To determine bass and melody notes a threshold was applied to the spectrogram by setting bounds to the frequency and setting a cutoff on the amplitude. In this slice of the spectrogram the dominant frequency was determined using a summation (integral), using the amplitude $A(f)$ as a weight and computing

$$\langle f \rangle = \frac{\int f \cdot A(f)df}{\int A(f)df} = \frac{\sum_n A_n \cdot f_n}{\sum_n A_n} \tag{4}$$

This frequency was compared to a library of musical notes and rounded to the nearest value. Averaging techniques were used to remove noise (usually due to the guitar pedals or specific playing methods like sliding) and a final output with note versus time was printed. Melody removal was attempted using a bandstop filter (an inverted Gaussian) and applying it to overtones (see the `filterNotes` method in Appendix B), but the spread of frequencies was hard to match to the filter and instead a simple second order low pass/high pass filter was used to separate the melody and bass.

3 Algorithm Implementation and Development

The Fast Fourier Transform (FFT) algorithm was used to Fourier transform the windowed data in the Gabor transform. The implementation in MATLAB uses the Cooley-Tukey method [1] that operates in $O(n \log n)$ runtime and requires shifting the zero frequency for manipulation in frequency. Generation of the spectrogram with the Gabor transform required applying the Gaussian window kernel at each time step. Since the music files contain millions of data points and the frequencies of interest were all less 10kHz a linear interpolation was applied to compress the data. In addition, the whole song was broken up into windows of equal width and the kernel was only applied to window pairs around a specified point. These two steps drastically increased the run time and lowered memory usage, and can be summarized by the following algorithm:

Algorithm 1: Gabor Filter algorithm

Create time window [0 ... W] where W = width
Create Gaussian filter in time window
Create time list [0 W/2 W 3W/2 ... Tend]
for each time in time list **do**
  grab a data centered at time
  if data window doesn't match Gaussian filter **then**
    Pad with zeros on each end
  end if
  multiply by Gaussian filter
  store FFT to spectrogram
  shift FFT, generate frequency list
end for
return spectrogram = complex matrix, times list, frequency list

Note isolation and determination was conducted on a truncated dataset, ignore all data outside of the frequency and amplitude range. Looping through each time, the expected frequency was calculated using Equation 4. This frequency was matched up with an imported library to determine note ID (i.e. C4 or A#5) based on the nearest frequency value. A final list of note changes and respective time values were used to reconstruct the song.

Bass and melody isolation were conducted on *Comfortably Numb* using a simple second order filter. This filter was applied on the frequency at all time steps using a high pass and low pass filter of the form

$$F_{LP} = \frac{1}{\left(1 + \frac{if}{f_0}\right)^2} \text{ and } F_{HP} = \left(\frac{\frac{f}{f_0}}{1 + \frac{if}{f_0}}\right)^2 \tag{5}$$

respectively, where f_0 is the knee frequency. This knee frequency was determined as the highest frequency of the bass (low pass filter applied) and the lowest frequency of the melody (high pass filter applied).

Recreating a music score after filtering the melody and bass required a more rigorous process due to the tendency for overtones and noise to shift the measured frequency up and down between timesteps. To mitigate this, a sliding window was used to smooth the score outlined below:

Algorithm 2: Note Smoothing Algorithm

Create a list of indices for each time point
for each time in time list **do**
  Apply amplitude threshold to frequency spectrum
  Find amplitude weighted average frequency
  if no points above threshold **then**
    Store previous time note index
  else
    Store index of closest note from note list
  end if
end for
Create another list of indices for each time point
for each index **do**
  Get n nearest neighbor indices
  Find most common index among neighbors, store to second list
end for
return second list of indices

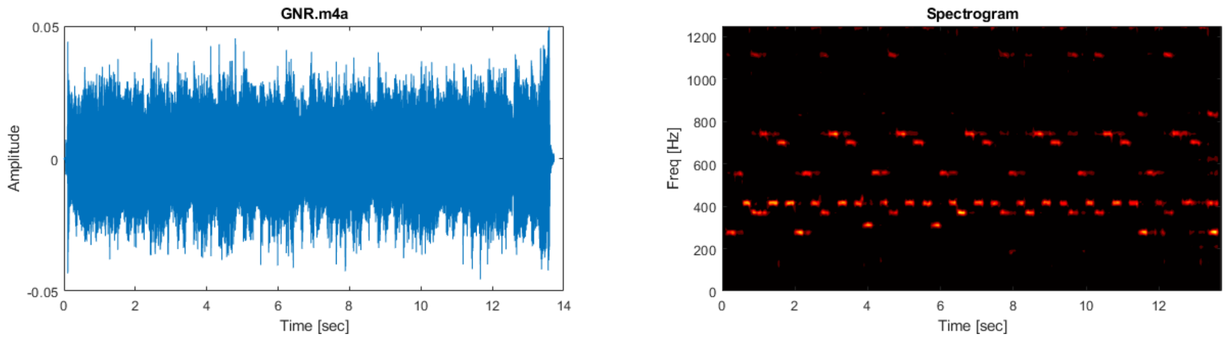

Figure 2: The original (compressed) audio signal from `GNR.m4a` (left) and the spectrogram generated using a window width of 0.05 seconds (right)

This useful for centering notes that were at the edge of threshold and lower the overall number of notes written to the music score.

4 Computational Results

The Gabor transform was first applied to the *Sweet Child O'Mine* clip, `GNR.m4a`, to get the results shown below in Figure 2. Since this opening only had one instrument it was possible to extract note information without any kind of filtering or manipulation.

Following is a table with the first 20 notes of the clip and times (a representation of the music score) with errors marked in red:

Time (sec)	0.125	0.575	0.825	1.075	1.15	1.2	1.275	1.525	1.775	2.025
Note	C#4	G#4	F#4	F#5	F#4	F#5	G#4	F5	G#4	C#4

2.225	2.475	2.75	2.95	3.2	3.425	3.675	3.825	3.95	4.15	4.45
C#5	G#4	F#4	F#5	G#4	F5	G#4	A4	D#4	C#5	G#4

The first two errors marked here (and most other errors found in the melody isolation) were octave errors likely caused by the dominance of an overtone during that sample. The third error was an extra note added due to the expected frequency shifting upwards. No further filtering of data was conducted after the Gabor transform and only frequency/amplitude thresholds were applied to the data. To see a list of all notes extracted, refer to Appendix C.

Analysis of *Comfortably Numb* (the clip titled `Floyd.m4a`) was conducted by separating the melody and bass using the second order filter shown in Equation 5. The spectrograms show a clear distinction as shown in Figure 3, with the knee frequency set to 150Hz for the bass and 250Hz for the melody. The notes in the melody and bass were both determined and both showed significantly more error than the `GNR.m4a` clip because of the nature of the instruments before filtering. This was especially true for the melody, as the guitar playing used sliding and string bending causing abrupt changes and creating an "autotune" effect during note transcription. After applying a sliding window filter (that took the mode of the set of notes in that window), the results are shown below. The first 20 notes of the bass score were found to be

Time (sec)	0.275	1.075	2.325	2.900	6.275	6.650	6.675	7.075	7.675	8.025
Note	B2	A#2	B2	A#2	A2	G2	C3	F#2	G2	F#2

8.125	8.600	8.725	9.050	9.075	9.300	9.800	9.850	10.025	10.075
G2	F2	D#2	A#2	A2	G2	D#2	G2	D#2	F2

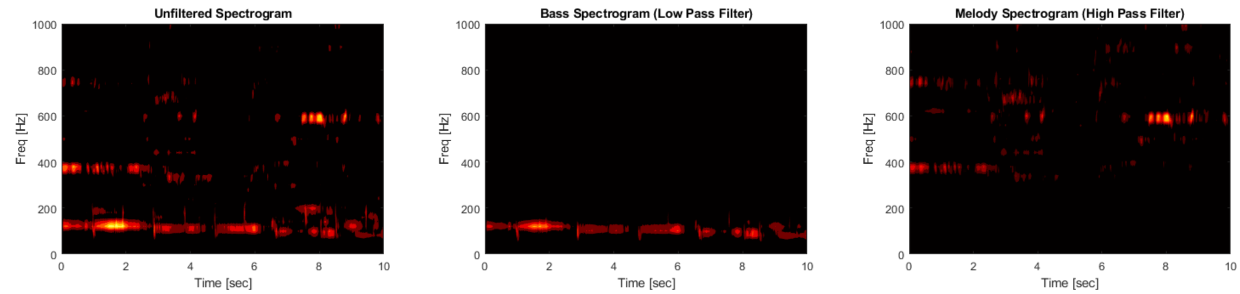

Figure 3: The first 10 seconds of `Floyd.m4a` converted to an unfiltered spectrogram (left) followed by the low-pass filtered (middle) and high-pass filtered (right) spectrograms

with errors highlighted in red. Similarly, the first 20 notes of the melody transcription were found to be

Time (sec)	0.300	0.650	0.700	0.725	0.750	0.775	0.900	1.250	1.325
Note	G#4	G4	G#4	A4	G#4	A4	G4	F#4	G4

1.350	1.425	1.450	1.475	1.925	1.950	1.975	2.000	2.150	2.225	2.450
G#4	G4	F#4	G4	F#4	C#5	A#4	C#5	G4	G#4	G4

with errors again highlighted in red. Note that this duration had a single guitar note but the tremolo/string bending caused the note to waver between F# and G, so those are counted here as correct results. The window used for bass collection was 20-200Hz and the window for melody collection was 200-600Hz. The melody window was selected to separated overtones from the main melody, although the solo spans more than an octave. An attempt was made to remove overtones (see the `filterNotes` method in Appendix B) The full music score for the base and first 100 transcribed notes of the melody are shown Appendix C (errors are not labeled, although there are many).

5 Summary and Conclusions

The implemented Gabor transform used for spectral analysis was quite efficient and low memory allowing for accurate visualization of the songs and the ability to manipulate the frequency and time data. Even from the contour plot of the spectrogram it was possible to identify most of the notes and timing and from this spectrogram it was straight forward to apply filtering to separate different frequency ranges for analysis. Initially there were issues with preservation of the complex coefficients but the transform was fully reversible after some manipulation of the algorithm.

Note identification using filtering techniques was a much harder problem and not completely solved in this project. The `GNR.m4a` clip had a single clear and steady melody, making it quite straight forward to track and convert to a music score. However, the `Floyd.m4a` clip had multiple instruments, making it much harder to isolate the notes and remove overtones. Overtones played much more of a role (they are apparent even on the spectrogram contour plot) leading to issues with finding the weighted average of the spectrum. For the bass score, the use of averaging the note over a window improved the accuracy of the algorithm but this lead to problems with the melody that was much faster and not always exactly in tune.

Some methods that could be used to improve this (and were not attempted here) would be to use a wider window sample size with the Gabor transform to get better precision on the expected frequency at each time. In addition, simply integrating each frequency to get an expected value was not enough for determining the score as there were likely multiple peaks in each frequency range. A peak finding algorithm that could be applied at each time step and could handle simultaneous notes would greatly improve this part of the analysis.

References

[1] James W. Cooley and John W. Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series". In: *Mathematics of Computation* 19.90 (1965), pp. 297–301. ISSN: 00255718, 10886842. URL: `http://www.jstor.org/stable/2003354`.

[2] I-Hui Hsieh and Kourosh Saberi. "Imperfect pitch: Gabor's uncertainty principle and the pitch of extremely brief sounds". In: *Psychonomic bulletin & review* 23 (May 2015). DOI: `10.3758/s13423-015-0863-y`.

[3] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

Appendix A MATLAB Functions

The MATLAB functions used for these calculations are listed below:

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `y = padarray(x, n)` pads vector `x` with n zeros distributed evenly at the beginning and end
- `contourf(x,y,Z)` generates a contour plot to visualize the matrix `Z` with columns spanning the vector `x` and rows spanning the vector `y`
- `Y = fftshift(X)` shifts the Fourier Transform X along each dimension, moving the zero-frequency to the center.
- `p = audioplayer(y, Fs)` creates an audio player object based on the amplitude vector `y` and sampling frequency `Fs` that can be played using `playblocking(p)`

Functions used for basic arithmetic or plot formatting are not included above, but the important figures are included in the Computation Results section above.

Appendix B MATLAB Code

All code is located in the "Homework 2" folder of the github repository (https://github.com/wliverno/AMATH582).

Several MATLAB files were used for this method, the first being `getSpectrogram.m` which generates a spectrogram based on the input file name, window width, and maximum frequency:

```
function [spectrogram, tg, freq] = getSpectrogram(inputFile, width, maxFreq)
    %Default Inputs:
    %inputFile ='GNR.m4a'
    %width=0.1; % window width (sec)
    %maxFreq=12000; %Hertz

    [y, Fs] = audioread(inputFile);
    tEnd = length(y)/Fs;
    t =(1:length(y))/Fs;

    %Compress file to cutoff higher frequency terms
    maxFreq = maxFreq - rem(maxFreq, 1/width);
    tc = linspace(0,tEnd,maxFreq*tEnd);
    yc = interp1(t, y, tc);
    yc(1) = 0;

    %Plot time domain signal
    subplot(1,2,1), plot(tc,yc);
    xlabel('Time [sec]');
    ylabel('Amplitude');
    title(inputFile);

    %Gabor Transform Lists
    tg = 0:width/2:tEnd;
    n = width*maxFreq;
    tw = linspace(0,width,n);
    freqs = (1/width)*[0:(n/2 - 1) -n/2:-1]; freq=fftshift(freqs);
    spectrogram = zeros(n,length(tg));
    filt = exp(-10*((tw/width)-0.5).^2);

    %Apply Transform across windows
    for i=2:(length(tg)-1)
        firstInd = round(tg(i-1)*maxFreq+1);
        lastInd = round(tg(i+1)*maxFreq);
        ind = firstInd:lastInd;
        if length(ind)<n
            y_= padarray(yc(ind), n-length(ind));
        else
            y_=yc(ind);
        end
        yf = y_.*filt;
        spectrogram(:,i) = ifftshift(fft(yf));
    end


    %Plot Results and play song
    subplot(1,2,2), contourf(tg, freq, abs(spectrogram), 'LineStyle', 'none'), colormap(hot);
    xlabel('Time [sec]');
```

```
    ylabel('Freq [Hz]');
    title('Spectrogram');
    axis([0 tEnd 0 maxFreq/8])

    %Reverse Spectrogram and check audio/graph for debug
    [maxFreq, yback] = reverseSpectrogram(spectrogram,tg,freq);
    tback =(1:length(yback))/maxFreq;
    %subplot(2,1,1), plot(tback, yback);
    %p8 = audioplayer(yback(1:length(yback)/10),maxFreq);
    %playblocking(p8);
end
```

The file `reverseSpectrogram.m` converts the spectrogram generated above back to a 1-D vector in the time domain:

```
function [Fs, data]= reverseSpectrogram(spectrogram, tg, freq)
    n = length(freq);
    width = tg(2);
    Fs = n/(2*width);
    tw = linspace(0,width,n);
    filt = exp(-10*((tw/width)-0.5).^2);
    data=zeros(floor(tg(end)*Fs),1);
    for i=2:(length(tg)-1)
        firstInd = round(tg(i-1)*Fs+1);
        lastInd = round(tg(i+1)*Fs);
        ind = firstInd:lastInd;
        data(ind) = ifft(fftshift(spectrogram(:,i)))./filt';
    end
end
```

The file `getNotes.m` was used to generate the score for the `GNR.m4a` audio clip:

```
function out = getNotes(file, upperThres, lowerThres)
    %Load library
    load notedata.mat
    %Load spectrogram
    [spec, tg, freq] = getSpectrogram(file,0.05, 10000);
    %Loop through and extract notes
    notes = ["Begin"];
    times = [0];
    freqs = zeros(1, length(tg));
    inds = zeros(1, length(tg));
    for i=1:length(tg)
        ft = abs(spec(freq>lowerThres & freq<upperThres,i))';
        ft = ft.*(ft>1);
        f = freq(freq>lowerThres & freq<upperThres);
        expFreq = sum(ft.*f)/sum(ft);
        [minValue,ind] = min(abs(noteFreq-expFreq));
        freqs(i) = expFreq;
        inds(i)=ind;
        if (isnan(expFreq) | expFreq == 0) & i>1
            freqs(i) = freqs(i-1);
            inds(i)=inds(i-1);
        else
            freqs(i) = expFreq;
            inds(i)=ind;
```

```
        end

        if (notes(end)~=noteStrings(ind)) & (expFreq<upperThres)
            times = [times, tg(i)];
            notes = [notes, noteStrings(ind)];
        end
    end
    %Debugging: Plot note frequencies
    figure;
    plot(tg,noteFreq(inds))
    axis([0 tg(end) lowerThres upperThres])
    out = [string(times)', notes']
end
```

The file `floydAnalysis.m` was used to separate the melody/bass and generate the score for the `Floyd.m4a` audio clip:

```
clear all, close all;

load notedata.mat

[spec, tg, freq] = getSpectrogram('Floyd.m4a',0.05, 20000);
n = length(freq);
width = tg(2);
Fs = n/(2*width);

%An attempt at removing melody + overtones (not used)
%spec = filterNotes(spec, freq, tg, 450, 250);
%spec = filterNotes(spec, freq, tg, 550, 300);
%spec = filterNotes(spec, freq, tg, 700, 400);
%spec = filterNotes(spec, freq, tg, 900, 500);
%spec = filterNotes(spec, freq, tg, 500, 400);

% [maxFreq, yback] = reverseSpectrogram(spec,tg,freq);
% tback =(1:length(yback))/maxFreq;
% p8 = audioplayer(yback(1:length(yback)/8),maxFreq);
% playblocking(p8);

%Simple Low-Pass Filter
f0 = 150; %Hz
filt = 1./(1+(1i*freq/f0))'; %Positive Frequencies
filt = filt./(1-(1i*freq/f0))';  %Negative Frequencies
filt = filt.^2; %Poof - now it's a second order filter!
specbass = zeros(size(spec));
for i=1:length(tg)
    specbass(:, i) = spec(:, i).*filt;
end

%Simple High-Pass Filter
f0 = 250; %Hz
filt = (freq/f0)'./(1+(1i*freq/f0))'; %Positive Frequencies
filt = filt.*(freq/f0)'./(1-(1i*freq/f0))';  %Negative Frequencies
filt = 2*filt.^2; %Poof - now it's a second order filter!
specmelody = zeros(size(spec));
for i=1:length(tg)
```

```
    specmelody(:, i) = spec(:, i).*filt;
end

%Plot spectrograms
figure;
normSpec = abs(spec);
normSpecBass = abs(specbass);
normSpecMelody = abs(specmelody);
subplot(1,3,1), contourf(tg, freq, normSpec, 'LineStyle', 'none'), colormap(hot);
xlabel('Time [sec]');
ylabel('Freq [Hz]');
title('Unfiltered Spectrogram');
axis([0 10 0 1000])
subplot(1,3,2), contourf(tg, freq, normSpecBass, 'LineStyle', 'none'), colormap(hot);
xlabel('Time [sec]');
ylabel('Freq [Hz]');
title('Bass Spectrogram (Low Pass Filter)');
axis([0 10 0 1000])
subplot(1,3,3), contourf(tg, freq, normSpecMelody, 'LineStyle', 'none'), colormap(hot);
xlabel('Time [sec]');
ylabel('Freq [Hz]');
title('Melody Spectrogram (High Pass Filter)');
axis([0 10 0 1000])

% Debugging - inspect resulting audio signal and listen to playback
[maxFreq, yback] = reverseSpectrogram(specmelody,tg,freq);
tback =(1:length(yback))/maxFreq;
%figure;
%subplot(2,1,1), plot(tback, yback);
%p8 = audioplayer(yback(1:length(yback)/8),maxFreq);
%playblocking(p8);

[bassNotes, bassNoteTimes] = getNotesSpec(normSpecBass, freq, tg, 150, 20, 10)
[melodyNotes, melodyNoteTimes] = getNotesSpec(normSpecMelody, freq, tg, 600, 200, 5)

%Method for getting notes from filtered spectrum
function [notes, times] = getNotesSpec(normSpec, freq, tg, upper, lower, smoothfactor)
    load notedata.mat
    inds = zeros(1, length(tg));
    for i=1:length(tg)
        ft = normSpec(freq>lower & freq < upper,i)'.*(normSpec(freq>lower & freq < upper,i)'>1);
        f = freq(freq>lower & freq < upper);
        expFreq = sum(ft.*f)/sum(ft);
        [minValue,ind] = min(abs(noteFreq-expFreq));
        if expFreq > 0
            inds(i) = ind;
        elseif i>1
            inds(i) = inds(i-1);
        else
            inds(i) = ind;
        end
    end
    %Smooth results using the mode
    indsFilt = zeros(1, length(tg));
```

```
    notes = ["Begin"];
    times = [0];
    for i=1:length(tg)
        if i<=smoothfactor/2
            indsFilt(i) = mode(inds(1:smoothfactor));
        elseif i>=length(tg) - smoothfactor/2
            indsFilt(i) = mode(inds(length(tg)-smoothfactor:end));
        else
            indsFilt(i) = mode(inds(i-floor(smoothfactor/2):i+floor(smoothfactor/2)));
        end
        currentNote = noteStrings(indsFilt(i));
        if (notes(end)~=currentNote)
            times = [times, tg(i)];
            notes = [notes, noteStrings(indsFilt(i))];
        end
    end
    figure;
    plot(tg, noteFreq(indsFilt));
end

%Unused method for removing melody and overtones
function spec = filterNotes(spec, freq, tg, upperThreshFreq, lowerThreshFreq)
    rng = upperThreshFreq-lowerThreshFreq;
    [foo,indUp] = min(abs(freq-upperThreshFreq));
    [foo,indLo] = min(abs(freq-lowerThreshFreq));
    for i=1:length(tg)
        ft = abs(spec(indLo:indUp,i))'.*(abs(spec(indLo:indUp,i))'>0.7);
        %spec(indLo:indUp,i) = spec(indLo:indUp,i).*~ft';
        f = freq(indLo:indUp);
        expFreq = sum(ft.*f)/sum(ft);
        if ~isnan(expFreq)
            filter = ones(1, length(freq));
            for j=1:3
                filter = filter - exp(-1*(freq - (j*expFreq)).^2/(1000));
                filter = filter - exp(-1*(freq + (j*expFreq)).^2/(1000));
            end
            %filter = filter.^4;
            spec(:,i) = real(spec(:,i)).*filter' + (1i*imag(spec(:,i)));
        else
            warning('Frequency Omitted');
        end
    end
end
```

Appendix C Music Scores

`GNR.m4a` **melody:**

===========
Time - Note
===========
0.125 - C#4/Db4
0.575 - G#4/Ab4
0.825 - F#4/Gb4

1.075 - F#5/Gb5
1.150 - F#4/Gb4
1.200 - F#5/Gb5
1.275 - G#4/Ab4
1.525 - F5
1.775 - G#4/Ab4
2.025 - C#4/Db4
2.225 - C#5/Db5
2.475 - G#4/Ab4
2.750 - F#4/Gb4
2.950 - F#5/Gb5
3.200 - G#4/Ab4
3.425 - F5
3.675 - G#4/Ab4
3.825 - A4
3.950 - D#4/Eb4
4.150 - C#5/Db5
4.450 - G#4/Ab4
4.825 - F#5/Gb5
5.050 - G#4/Ab4
5.275 - F5
5.550 - G#4/Ab4
5.650 - G4
5.675 - G#4/Ab4
5.850 - D#4/Eb4
6.025 - C#5/Db5
6.250 - G#4/Ab4
6.475 - F#4/Gb4
6.700 - F#5/Gb5
6.950 - G#4/Ab4
7.175 - F5
7.425 - G#4/Ab4
7.700 - F#4/Gb4
8.150 - G#4/Ab4
8.375 - F#4/Gb4
8.575 - F#5/Gb5
8.825 - G#4/Ab4
9.050 - F5
9.300 - G#4/Ab4
9.575 - F#4/Gb4
9.800 - C#5/Db5
10.025 - G#4/Ab4
10.300 - F#4/Gb4
10.475 - F#5/Gb5
10.750 - G#4/Ab4
10.975 - F5
11.200 - G#4/Ab4
11.500 - C#4/Db4
11.725 - C#5/Db5
12.125 - C#4/Db4
12.400 - F#5/Gb5
12.750 - G#4/Ab4
12.900 - C#5/Db5
12.925 - F5

13.375 - F4
13.475 - C#4/Db4
13.575 - E4
13.600 - F4

`Floyd.m4a` **bass:**

===========
Time - Note
===========
0.000 - Begin
0.000 - B2
0.275 - A#2/Bb2
1.075 - B2
2.325 - A#2/Bb2
2.900 - A2
6.275 - G2
6.650 - C3
6.675 - F#2/Gb2
7.075 - G2
7.675 - F#2/Gb2
8.025 - G2
8.125 - F2
8.600 - D#2/Eb2
8.725 - A#2/Bb2
9.050 - A2
9.075 - G2
9.300 - D#2/Eb2
9.800 - G2
9.850 - D#2/Eb2
10.025 - F2
10.075 - D#2/Eb2
10.450 - B2
10.750 - A#2/Bb2
11.050 - B2
11.250 - A#2/Bb2
12.325 - F#2/Gb2
12.525 - A#2/Bb2
13.250 - C3
13.475 - A#2/Bb2
13.525 - B2
13.550 - A#2/Bb2
13.950 - C3
14.200 - B2
14.225 - A#2/Bb2
14.250 - B2
14.475 - A#2/Bb2
16.175 - B2
16.575 - A#2/Bb2
18.025 - A2
19.025 - G2
19.050 - A2
19.075 - G2
19.200 - A2
19.875 - D#2/Eb2

20.150 - G2
20.350 - C#2/Db2
20.550 - A2
21.725 - F2
21.775 - F#2/Gb2
22.025 - G2
22.800 - F#2/Gb2
23.075 - G2
23.225 - F2
23.700 - G2
23.800 - F#2/Gb2
23.875 - F2
24.050 - D#2/Eb2
24.350 - A#2/Bb2
24.925 - D#2/Eb2
25.550 - G2
25.575 - A#2/Bb2
25.850 - B2
26.125 - A#2/Bb2
28.925 - A2
29.125 - A#2/Bb2
32.400 - A2
32.575 - A#2/Bb2
33.100 - A2
34.500 - C#2/Db2
34.750 - A2
34.975 - D#2/Eb2
35.150 - F2
35.175 - C#3/Db3
35.475 - A2
35.925 - A#2/Bb2
36.200 - F#2/Gb2
36.225 - G2
36.650 - D#2/Eb2
36.850 - F#2/Gb2
37.350 - G2
37.825 - F#2/Gb2
38.275 - F2
38.775 - A#2/Bb2
39.025 - A2
39.075 - G2
39.275 - F#2/Gb2
39.400 - D#2/Eb2
40.000 - G2
40.175 - G#2/Ab2
40.250 - F2
40.475 - A#2/Bb2
40.500 - G2
40.575 - A#2/Bb2
42.600 - B2
43.100 - A#2/Bb2
43.275 - B2
43.350 - A#2/Bb2
43.375 - B2

43.400 - A#2/Bb2
43.975 - A2
44.300 - G#2/Ab2
44.325 - A2
44.425 - A#2/Bb2
44.625 - B2
44.925 - A#2/Bb2
46.875 - B2
47.100 - A#2/Bb2
47.250 - G2
47.450 - A#2/Bb2
48.525 - A2
48.550 - G2
48.900 - A2
49.075 - G2
49.675 - A2
51.700 - G2
52.025 - F#2/Gb2
52.675 - G2
53.150 - F#2/Gb2
53.425 - F2
53.925 - A#2/Bb2
53.950 - G#2/Ab2
54.125 - F#2/Gb2
54.500 - F2
54.725 - D#2/Eb2
54.950 - F2
54.975 - D#2/Eb2
55.300 - F2
55.350 - D#2/Eb2
55.375 - F2
55.400 - D#2/Eb2
55.525 - F2
55.575 - G2
55.725 - F#2/Gb2
55.900 - A#2/Bb2
56.775 - B2
56.850 - A#2/Bb2
56.925 - B2
57.725 - A#2/Bb2
58.025 - B2
58.075 - A#2/Bb2
58.250 - A2
58.850 - G2
58.950 - A2
58.975 - G2
59.050 - G#2/Ab2
59.175 - A2
59.200 - G#2/Ab2
59.250 - A#2/Bb2

`Floyd.m4a` **melody (first 100 notes only):**

==========
Time - Note

===========
0.000 - G#4/Ab4
0.300 - G4
0.650 - G#4/Ab4
0.700 - A4
0.725 - G#4/Ab4
0.750 - A4
0.775 - G4
0.900 - F#4/Gb4
1.250 - G4
1.325 - G#4/Ab4
1.350 - G4
1.425 - F#4/Gb4
1.450 - G4
1.475 - F#4/Gb4
1.925 - C#5/Db5
1.950 - A#4/Bb4
1.975 - C#5/Db5
2.000 - G4
2.150 - G#4/Ab4
2.225 - G4
2.450 - F4
2.550 - A4
2.600 - A#4/Bb4
2.725 - G4
2.850 - G#4/Ab4
2.875 - A4
3.000 - A#4/Bb4
3.125 - G#4/Ab4
3.150 - G4
3.175 - G#4/Ab4
3.325 - F#4/Gb4
3.375 - G4
3.400 - F#4/Gb4
3.425 - G4
3.475 - G#4/Ab4
3.525 - G4
3.550 - G#4/Ab4
3.575 - G4
3.625 - A#4/Bb4
3.650 - A4
3.725 - G#4/Ab4
3.750 - A4
3.775 - G#4/Ab4
3.825 - B4
3.850 - A#4/Bb4
4.025 - B4
4.050 - A#4/Bb4
4.200 - G4
4.325 - G#4/Ab4
4.425 - F#4/Gb4
4.500 - G4
4.525 - E4
4.750 - A#4/Bb4

4.925 - G4
5.000 - A#4/Bb4
5.075 - B4
5.150 - C5
5.200 - F#4/Gb4
5.275 - C#5/Db5
5.325 - B4
5.375 - F4
5.400 - E4
5.675 - A#4/Bb4
5.775 - A4
5.875 - E4
5.925 - G4
6.050 - F#4/Gb4
6.125 - G#4/Ab4
6.250 - G4
6.300 - F4
6.325 - F#4/Gb4
6.475 - G4
6.500 - A4
6.575 - G#4/Ab4
6.700 - B4
6.725 - A#4/Bb4
6.750 - B4
6.775 - A4
6.850 - G#4/Ab4
6.875 - A4
6.900 - F#4/Gb4
6.950 - A4
7.225 - A#4/Bb4
7.350 - C5
7.425 - C#5/Db5
7.575 - C5
7.775 - C#5/Db5
7.925 - C5
7.950 - C#5/Db5
7.975 - C5
8.125 - C#5/Db5
8.150 - C5
8.175 - C#5/Db5
8.225 - C5
8.275 - B4
8.325 - A#4/Bb4
8.375 - C5
8.425 - A#4/Bb4
8.450 - C5
8.675 - B4